Task 3 - CRUD operations with IndexedDB and a RESTful API for Sedibelo Technologies.

In this document, I will provide a detailed explanation of the technical approach, design decisions, and implementation strategies I used to develop secure CRUD operations with IndexedDB and a RESTful API for Sedibelo Technologies. My focus throughout was on security, performance, scalability, and reliability—ensuring that the system meets modern development standards and can handle real-world challenges.

The use of IndexedDB for local client-side storage is an effective way to store structured data offline, and I integrated encryption using the Web Crypto API to ensure sensitive information remains protected. Furthermore, I built a RESTful API with Node.js to enable secure communication between the frontend and backend, providing seamless data management.

This solution demonstrates my ability to design and implement secure, efficient, and scalable CRUD operations using IndexedDB for local storage and Node.js with Express for backend simulation. The Web Crypto API ensures that all sensitive data is encrypted, and session tokens provide secure communication with the backend. I focused on reliability, idempotency, and performance, ensuring that the system can handle real-world challenges effectively.

This project reflects my expertise in managing encrypted data, building RESTful APIs, and optimizing performance for modern web applications.

Task 3.1 - Initializing IndexedDB for Secure Data Storage

The first step involved setting up IndexedDB to handle the secure storage of user data on the client side. IndexedDB is ideal for this purpose due to its ability to store large datasets locally in a structured format.I structured the IndexedDB database with an object store for user records, using UUIDs (Universally Unique Identifiers) as unique keys for each entry.

UUIDs ensured that all records remain globally unique, even if duplicated across distributed systems. To maintain smooth user interactions, I ensured that all operations—such as reads and writes—are asynchronous, preventing the application from freezing or becoming unresponsive.

Using IndexedDB also aligns with modern web standards because it enables the application to function offline while keeping data synchronized when back online.

The asynchronous nature of IndexedDB is a core feature that improves frontend performance by preventing blocking operations (Mozilla, 2023). Its design supports complex key-value pair storage, which makes it an excellent tool for handling structured data in progressive web apps.

Task 3.2 - Encrypting Data Using the Web Crypto API

Given that the data stored in IndexedDB may contain sensitive information, my next priority was to encrypt the data before saving it. This step ensures that even if unauthorized access to the IndexedDB database occurs, the stored data will remain unreadable.

I used the AES-GCM (Advanced Encryption Standard with Galois/Counter Mode) for encryption, as it provides both confidentiality and data integrity. AES-GCM is highly efficient and widely recommended for secure web applications due to its minimal overhead and built-in message authentication.

To generate encryption keys securely, I relied on password-based encryption where the key is derived from a user-provided password through PBKDF2 (Password-Based Key Derivation Function 2). This approach eliminates the risk of hardcoding keys into the system. Additionally, each encryption operation uses a unique Initialization Vector (IV) to ensure that even identical data will have distinct encrypted outputs.

AES-GCM is favored in modern web encryption because it provides fast and secure encryption with authentication (NIST, 2021). Using PBKDF2 for key derivation enhances security by adding a salt and iterating the hashing function, which mitigates brute-force attacks.

Task 3.3 - Implementing CRUD Operations

The next step involved developing the full range of Create, Read, Update, and Delete (CRUD) operations for managing encrypted data within IndexedDB. The goal was to ensure data security while enabling smooth user interactions.

Technical Approach:

- Create: I encrypt user data using AES-GCM and store it in IndexedDB with a UUID as the key.
- Read: When fetching data, I decrypt it on the fly, ensuring that decrypted information is only available temporarily in memory.
- Update: I retrieve, decrypt, modify, and re-encrypt existing records to maintain data consistency.
- Delete: Records are removed based on their UUID, ensuring that outdated or irrelevant data is securely deleted.

I designed these operations to be idempotent—repeated requests will produce the same outcome, ensuring reliability in case of network or system failures. This ensures that even if a user initiates multiple updates or deletes, the database remains consistent and without duplicate entries.Idempotency is a crucial principle in modern web design, especially for APIs and database operations (Fielding, 2000). It ensures system stability and consistency, particularly when handling unpredictable user behavior or network disruptions.

Task 3.4 - RESTful API Implementation with Node.js

To enable secure communication between the client-side IndexedDB storage and the backend, I developed a RESTful API using Node.js and Express. The API facilitates data synchronization, ensuring that the frontend and backend can operate in tandem.

Technical Approach:

- Endpoints:
    - /uniqueUsers: Retrieves all unique users stored in the IndexedDB database.
    - /addUser: Adds a new user entry to the database.
    - /updateUser: Updates an existing user’s details.
    - /deleteUser: Removes a user based on their UUID.

I ensured that all API calls are authenticated using session tokens to prevent unauthorized access. Each request includes a session token generated upon login, verifying the user's identity.

Additionally, I incorporated retry logic to handle network issues, ensuring that failed requests are retried a certain number of times before giving up. This ensures a robust and resilient system capable of handling intermittent connectivity issues.

RESTful APIs are a standard for communication in modern applications due to their scalability and stateless design (Fielding, 2000). Using session tokens for authentication ensures that API interactions remain secure and protected from unauthorized access.

Task 3.5 - Security Considerations

My primary concern throughout this task was ensuring that all stored and transmitted data remains secure. I integrated several layers of security to safeguard user data.

Technical Approach:

- Encryption: All data in IndexedDB is encrypted using AES-GCM, ensuring confidentiality and integrity.
- Secure Communication: The RESTful API requires session tokens for all operations to prevent unauthorized access.
- Unique IVs for Encryption: Each encryption uses a new Initialization Vector to prevent ciphertext reuse.
- Idempotent Operations: The CRUD operations were designed to ensure consistency and stability, even with repeated requests.

These measures ensure that the system adheres to modern security best practices and remains resilient against common attack vectors like unauthorized access and brute-force attacks.

Literature Reference:
The use of AES-GCM encryption and session-based authentication aligns with OWASP's recommendations for secure storage and communication (OWASP, 2024).

Task 3.6 - Performance and Scalability Considerations

I optimized the system to handle large datasets and multiple operations simultaneously without sacrificing performance.

Technical Approach:

- IndexedDB for Local Storage: IndexedDB's large storage capacity makes it suitable for storing extensive datasets offline.
- Batch Processing: I grouped operations into batches to reduce overhead and improve performance.
- Lazy Decryption: Data is decrypted only when required, minimizing the impact on memory usage.

These optimizations ensure that the system performs efficiently under various conditions, even with large datasets or multiple simultaneous operations.